

Pruning Algorithms for Pretropisms of Newton Polytopes*

Jeff Sommars Jan Verschelde

University of Illinois at Chicago
Department of Mathematics, Statistics, and Computer Science
851 S. Morgan Street (m/c 249), Chicago, IL 60607-7045, USA
{sommars1,janv}@uic.edu

Abstract

Pretropisms are candidates for the leading exponents of Puiseux series that represent positive dimensional solution sets of polynomial systems. We propose a new algorithm to both horizontally and vertically prune the tree of edges of a tuple of Newton polytopes. We provide experimental results with our preliminary implementation in Sage that demonstrates that our algorithm compares favorably to the definitional algorithm.

1 Introduction

Almost all polynomial systems arising in applications are sparse, as few monomials appear with nonzero coefficients, relative to the degree of the polynomials. Polyhedral methods exploit the sparse structure of a polynomial system. In the application of polyhedral methods to compute positive dimensional solution sets of polynomial systems, we look for series developments of the solutions, and in particular we look for Puiseux series [29]. The leading exponents of Puiseux series are called *tropisms*. The *Newton polytope* of a polynomial in several variables is the convex hull of the exponent tuples of the monomials that appear with nonzero coefficient in the polynomial.

In [10], polyhedral methods were defined in tropical algebraic geometry. We refer to [27] for a textbook introduction to tropical algebraic geometry. Our textbook reference for definitions and terminology of polytopes is [39].

Our problem involves the intersection of polyhedral cones. A *normal cone* of a face F of a polytope P is the convex cone generated by all of the facet normals of facets which contain F . The *normal fan* of a polytope P is the union of all of

*This material is based upon work supported by the National Science Foundation under Grant No. 1440534.

the normal cones of every face of P . Given two fans F_1 and F_2 , their *common refinement* $F_1 \wedge F_2$ is defined as

$$F_1 \wedge F_2 = \bigcup_{\substack{C_1 \in F_1 \\ C_2 \in F_2}} C_1 \cap C_2. \quad (1)$$

As the common refinement of two fans is again a fan, the common refinement of three fans F_1 , F_2 , and F_3 may be computed as $(F_1 \wedge F_2) \wedge F_3$.

Problem Statement. Given the normal fans (F_1, F_2, \dots, F_n) of the Newton polytopes (P_1, P_2, \dots, P_n) , a *pretropism* is a ray in a cone C ,

$$C = C_1 \cap C_2 \cap \dots \cap C_n \in F_1 \wedge F_2 \wedge \dots \wedge F_n, \quad (2)$$

where each C_i is the normal cone to some k_i -dimensional face of P_i , for $k_i \geq 1$, for $i = 1, 2, \dots, n$. Our problem can thus be stated as follows: given a tuple of Newton polytopes, compute all pretropisms.

We say that two pretropisms are equivalent if they are both perpendicular to the same tuples of faces of the Newton polytopes. Modulo this equivalence, there are only a finite number of pretropisms. Ours is a difficult problem because of the dimension restrictions on the cones. In particular, the number of pretropisms can be very small compared to the total number of cones in the common refinement.

Pretropisms are candidates tropisms, but not every pretropism is a tropism, as pretropisms depend only on the Newton polytopes of the system. For polynomial systems with sufficiently generic coefficients, every tropism is also a pretropism. See [9] for an example.

Related Work. A tropical prevariety was introduced in [10] and Gfan [26] is a software system to compute the common refinement of the normal fans of the Newton polytopes. Gfan relies on the reverse search algorithms [4] in cddlib [20].

The problem considered in this paper is a generalization of the problem to compute the mixed volume of a tuple of Newton polytopes, for which pruning methods were first proposed in [15]. Further developments can be found in [21] and [31], with corresponding free software packages MixedVol [22] and DEMiCS [30]. A recent parallel implementation along with a complexity study appears in [28]. The relationship between triangulations and the mixed subdivisions is explained and nicely illustrated in [13].

The main difference between mixed volume computation and the computation of the tropical prevariety is that in a mixed volume computation the vertices of the polytopes are lifted randomly, thus removing all degeneracies. This lifting gives the powers of an artificial parameter. In contrast, in a Puiseux series development of a space curve, the first variable is typically identified as the parameter and the powers of the first variable in the given polynomials cannot be considered as random.

A practical study on various software packages for exact volume computation of a polytope is described in [12]. Exact algorithms on Newton polytopes

are discussed in [18]. The authors of [16] present an experimental study of approximate polytope volume computation. In [17], a polynomial-time algorithm is presented to compute the edge skeleton of a polytope. Computing integer hulls of convex polytopes can be done with polymake [3].

Our contributions and organization of the paper. In this paper we outline two different types of pruning algorithms for the efficient computation of pretropisms. We report on a preliminary implementation in Sage [36] and illustrate the effectiveness on a parallel computer for various benchmark problems. This paper extends the results of our EuroCG paper [35] as well as [34].

2 Pruning Algorithms

2.1 Horizontal and Vertical Pruning Defined

Because we are interested only in those cones of the common refinement that contain rays perpendicular to faces of dimension one or higher, we work with the following modification of (1):

$$F_1 \wedge_1 F_2 = \bigcup_{\substack{C_1 \in F_1, C_1 \perp \text{ edge of } P_1 \\ C_2 \in F_2, C_2 \perp \text{ edge of } P_2}} C_1 \cap C_2. \quad (3)$$

The \wedge_1 defines the *vertical pruning* as the replacement of \wedge by \wedge_1 in $(F_1 \wedge F_2) \wedge F_3$ so we compute $(F_1 \wedge_1 F_2) \wedge_1 F_3$. Cones in the refinement that do not satisfy the dimension restrictions are pruned away in the computations. Our definition of vertical pruning is currently incomplete, but we will refine it in 2.5 after we have formally defined our algorithms.

The other type of pruning, called *horizontal pruning* is already partially implicitly present in the \bigcup operator of (3), as in a union of sets of cones, every cone is collected only once, even as it may originate as the result of many different cone intersections. With horizontal pruning we remove cones of $F_1 \wedge_1 F_2$ which are contained in larger cones. Formally, we can define this type of pruning via the \wedge_2 operator:

$$F_1 \wedge_2 F_2 = \bigcup_{\substack{C \in F_1 \wedge_1 F_2, C \not\subset C' \\ C' \in F_1 \wedge_1 F_2 \setminus \{C\}}} C. \quad (4)$$

2.2 Pseudo Code Definitions of the Algorithms

Algorithm 2 sketches the outline of our algorithm to compute all pretropisms of a set of n polytopes. Along the lines of the gift wrapping algorithm, for every edge of the first polytope we take the plane that contains this edge and consider where this plane touches the second polytope. Algorithm 1 starts exploring the edge skeleton defined by the edges connected to the vertices in this touching plane.

The exploration of the neighboring edges corresponds to tilting the ray r in Algorithm 1, as in rotating a hyperplane in the gift wrapping method. One may wonder why the exploration of the edge skeleton in Algorithm 1 needs to continue after the statement on line 4. This is because the cone C has the potential to intersect many cones in P , particularly if P has small cones and C is large. Furthermore it is reasonable to wonder why we bother checking cone containment when computing the intersection of two cones provides more useful information. Checking cone containment means checking if each of the generators of C is contained in C_E , which is a far less computationally expensive operation than computing the intersection of two cones.

In the Newton-Puiseux algorithm to compute series expansions, we are interested only in the edges on the lower hull of the Newton polytope, i.e. those edges that have an upward pointing inner normal. For Puiseux for space curves, the expansions are normalized so that the first exponent in the tropism is positive. Algorithm 2 is then easily adjusted so that calls to the edge skeleton computation of Algorithm 1 are made with rays that have a first component that is positive.

2.3 Correctness

To see that these algorithms will do what they claim, we must define an additional term. A *pretropism graph* is the set of edges for a polytope that have normal cones intersecting a given cone. We will now justify why the cones output by Algorithm 1 correspond to the full set of cones that live on a pretropism graph.

Theorem 2.1. *Pretropism graphs are connected graphs.*

Proof. Let C be a cone, and let P be a polytope with edges e_1, e_2 such that they are in the pretropism graph of C . Let C_1 be the cone of the intersection of the normal cone of e_1 with C , and let C_2 be the cone of the intersection of the normal cone of e_2 and C . If we can show that there exists a path between e_1 and e_2 that remains in the pretropism graph, then the result will follow.

Let n_1 be a normal to e_1 that is also in C_1 and let n_2 be a normal to e_2 that is also in C_2 . Set $n = tn_1 + (1 - t)n_2$ where $0 \leq t \leq 1$. Consider varying t from 0 to 1; this creates the cone C_n , a cone which must lie within C , as both n_1 and n_2 lie in that cone. As n moves from 0 to 1, it will progressively intersect new faces of P that have all of their edges in the pretropism graph. Eventually, this process terminates when we reach e_2 , and we have constructed a path from e_1 to e_2 . Since a path always exists, we can conclude that pretropism graphs are connected graphs. \square

Since pretropism graphs are connected, Algorithm 1 will find all cones of edges on the pretropism graph. In Algorithm 2, we iteratively explore the edge skeleton of polytope P_i , and use the pruned set of cones to explore P_{i+1} . From this, it is clear that Algorithm 2 will compute the full set of pretropisms.

2.4 Analysis of Computational Complexity

In estimating the cost of our algorithm to compute all pretropisms, we will first consider the case when there are two polytopes. We will take the primitive operation of computing pretropisms to be the number of cone intersections performed, as that number will drive the time required for the algorithm to complete. For a polytope P , denote by $n_e(P)$ its number of edges. The upper bound on the number of primitive operations for two polytopes P_1 and P_2 is the product $n_e(P_1) \times n_e(P_2)$, while the lower bound equals the number of pretropisms.

Denote by $E_{P,e}$ the pretropism graph resting on polytope P corresponding to the ray determined by edge e . Let $n_e(E_{P,e})$ denote the number of edges in $E_{P,e}$.

Proposition 2.2. *The number of primitive operations in Algorithm 2 on two polytopes P_1 and P_2 is bounded by*

$$\sum_{i=1}^{n_e(P_1)} n_e(E_{P_2, e_i}), \quad (5)$$

where e_i is the i -th edge of P_1 .

As $E_{P,e}$ is a subset of the edges of P : $n_e(E_{P,e}) \leq n_e(P)$. Therefore, the bound in (5) is smaller than $n_e(P_1) \times n_e(P_2)$.

To interpret (5), recall that Algorithm 2 takes a ray from inside a normal cone to an edge of the first polytope for the exploration of the edge graph of the second polytope. If we take a simplified view on the second polytopes as a ball, then shining a ray on that ball will illuminate at most half of its surface. If we use the estimate: $n_e(E_{P_2, e_i}) \approx n_e(P_2)/2$, then Algorithm 2 cuts the the upper bound on the number of primitive operations in half.

Estimating the cost of the case of n polytopes follows naturally from the cost analysis of the case of 2 polytopes. For n polytopes, the upper bound on the number of primitive operations required is the product $n_e(P_1) \times n_e(P_2) \times \dots \times n_e(P_n)$.

Proposition 2.3. *The number of primitive operations in Algorithm 2 on n polytopes P_1, P_2, \dots, P_n is bounded by*

$$\sum_{i=1}^{n_e(P_1)} \left(\prod_{j=2}^n n_e(E_{P_j, e_i}) \right) \quad (6)$$

where e_i is the i -th edge of P_1 .

Again, if we use the estimate that $n_e(E_{P_j, e_i}) \approx n_e(P_j)/2$, then Algorithm 2 reduces the upper bound on the number of primitive operations by $\frac{1}{2^{n-1}}$. This estimate depends entirely on the intuition that we are cutting the number of comparisons in half. This estimate may not hold in the case when we have large

lineality spaces, and thus have huge input cones. This situation can be partially remedied by sorting the input polytopes from smallest dimension of lineality space to highest. This seeds Algorithm 2 with the smallest possible input cones.

2.5 Horizontal and Vertical Pruning Revisited

The definitional algorithm of pretropism can be interpreted as creating a tree structure. From a root node, connect the cones of P_1 . On the next level of the tree, place the cones resulting from intersecting each of the cones of P_2 with the cones of P_1 , connecting the new cones with the cone from P_1 that they intersected. It is likely that at this level of the tree there are many cones that are empty. Continue this process creating new levels of cones representing the intersection of the previous level of cones with the next polytope until the n th polytope has been completed. The cones at the n th layer of the tree represent the cones generated by pretropisms.

Our algorithms can be seen to improve on this basic tree structure in two distinct ways. Algorithm 1 reduces the number of comparisons needed through exploring the edge skeletons of the polytopes. Because of this, there are many times that we do not perform cone intersections that will result in 0 dimensional cones. From the perspective of the tree, this is akin to avoiding drawing edges to many 0 dimensional cones; we call this vertically pruning the tree. We horizontally prune the tree through Algorithm 2 which reduces the number of cones necessary to follow for a given level. This is illustrated in Figure 1. By both horizontally and vertically pruning the tree of cones, we are able to avoid performing many unnecessary cone intersections. We will demonstrate the benefits of pruning experimentally in the sections to come.

3 Implementation

Our algorithm takes as input a modified version of the data structure output by the gift wrapping algorithm to compute convex hulls. It conceptually exploits the connectivity between vertices, edges, and facets, but only requires the edge skeleton of the polytope. To accomplish this, we created edge objects that had vertices, references to their neighboring edges, and the normal cone of the edge. When polytopes were not full dimensional, we included the generating rays of the lineality space when we created the normal cones. This has the negative effect of increasing the size of the cone, but is essential for the algorithm to work.

3.1 Code

We developed a high level version of Algorithm 2 in Sage [36], using its modules for lattice polytopes [32], and polyhedral cones [11]. To compute the intersections of cones, Sage uses PPL [6]. Our preliminary code is available at <https://github.com/sommars/GiftWrap>.

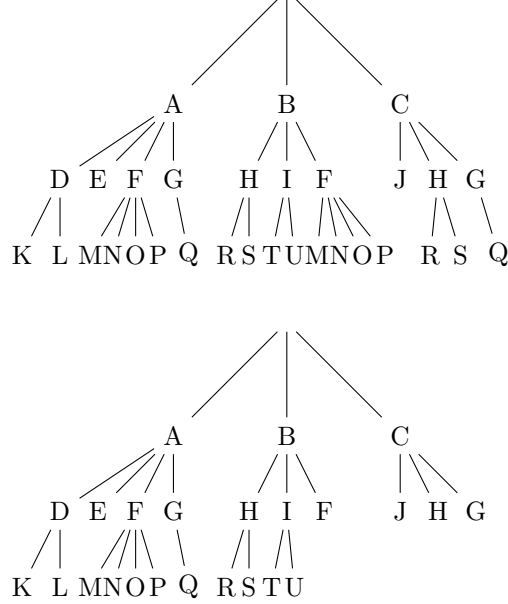


Figure 1: Nodes A, B, C represent cones to P_1 . Nodes D, E, F, and G represent intersections of cone A with cones to P_2 , etc. Nodes K and L represent intersections of cone D with cones to P_3 , etc. Duplicate nodes are removed from the second tree at the bottom.

3.2 Parallelism

To improve the performance of our core algorithms, we also implemented high level parallelism. We used the built in Python queue structure to create a job queue of cones in lines 5 through 7 of Algorithm 2. Each call to the Algorithm 1 is done independently on a distinct process, using the computers resources more efficiently.

Additionally, we have implemented parallelism in checking the cone containments in lines 8 through 12 of Algorithm 1. To check if cone C_1 is contained within another cone C_2 , it requires checking if each of the linear equations and inequalities of C_1 is or is not restricted by each of the linear equations and inequalities of C_2 . However, there is much overlap between cones, with many distinct cones sharing some of the same linear equations or inequalities. Because of this, we optimized by creating a lookup table to avoid performing duplicate calculations. However, with large benchmark problems, creation of this table becomes prohibitively slow. To amend this problem, we parallelized the creation of the table, with distinct processes performing distinct calculations.

4 Generic Experiments

To test the algorithms, we generate $n - 1$ simplices spanned by integer points with coordinates uniformly generated within the range of 0 to 30. This input corresponds to considering systems of $n - 1$ sparse Laurent polynomials in n variables with $n + 1$ monomials per equation. We can compare with the mixed volume computation if we add one extra linear equation to the Laurent polynomial system. Then the mixed volume of the n -tuple will give the sum of the degrees of all the curves represented by the Puiseux series. Assuming generic choices for the coefficients, the degrees of the curves can be computed directly from the tropisms, as used in [38] and applied in [1, 2].

Denoting by $MV(P)$ the mixed volume of an n -tuple P of Newton polytopes:

$$MV(P) = \sum_{\mathbf{v}} \left(\max_{i=1}^n v_i - \min \left(\min_{i=1}^n v_i, 0 \right) \right) \quad (7)$$

where the sum ranges over all tropisms \mathbf{v} .

All computations were done on a 2.6 GHz Intel Xeon E5-2670 processor in a Red Hat Linux workstation with 128 GB RAM using 32 threads. When performing generic tests, the program did not perform any cone containment tests because no cone can be contained in another cone in this case.

4.1 Benchmarking

Table 4.1 shows a comparison between the two distinct methods of computing pretropisms. The mixed volume was computed with the version of Mixed-Vol [22], available in PHCpack [37] since version 2.3.13. For systems with generic coefficients, the mixed volume equals the number of isolated solutions [7]. While a fast multicore workstation can compute millions of solutions, a true supercomputer will be needed in the case of billions of solutions. For larger dimensions, the new pruning method dominates the method suggested by the definition of pretropism.

Table 1: Comparisons between the definitional and our pruning method, for randomly generated generic simplices. Timings are listed in seconds.

n	Definitional	Pruning	#Pretropisms	Mixed Volume
3	0.008	0.20	7	319
4	0.11	0.42	18	7,384
5	1.33	0.76	58	152,054
6	13.03	2.75	171	4,305,758
7	243.88	20.17	614	91,381,325
8	2054.11	220.14	1,878	2,097,221,068

4.2 Number of Cone Intersections

Another way that the definitional algorithm can be compared to our new algorithm is through comparing the number of cone intersections required for each algorithm. Table 2 contains a comparison of these numbers. A large number of trials were performed at each dimension so we could conclude statistically if our mean number of intersections differed from the number of intersections required by the cone intersection algorithm. To test this hypothesis, we performed t -tests using the statistical software package R [33]. For every dimension from 3 to 8, we were able to reject the null hypothesis that they had the same mean and we were able to conclude that the new algorithm has a lower mean number of intersections ($p < 2 \times 10^{-16}$ for every test). We had estimated the cost to be an improvement by a factor of $\frac{1}{2^{n-1}}$, but experimentally we found a greater improvement as can be seen in Table 2.

Table 2: Average number of cone intersections required for each algorithm, comparing the definitional algorithm with our pruning algorithm for generic inputs. The second to last column contains the ratio predicted by our cost estimate and the final column contains the actual ratio.

n	Definitional	Pruning	Predicted Ratio	Actual Ratio
3	36	29	0.5	0.72
4	1,000	288	0.25	0.288
5	50,625	2,424	0.125	0.0478
6	4,084,101	18,479	0.0625	0.00452
7	481,890,304	145,134	0.03125	0.000301
8	78,364,164,096	1,150,386	0.015625	0.0000147

4.3 Comparison with Gfan

In the generic case, our code is competitive with Gfan. Table 3 contains timing comparisons, with input polynomials determined as they were previously determined; the timings in the Gfan column were obtained by running the current version 0.5 of Gfan [25].

5 Benchmark Polynomial Systems

Many of the classic mixed volume benchmark problems like Katsura- n , Chandra- n , eco- n , and Noonberg- n are inappropriate to use as benchmark systems for computing pretopisms. A good testing system needs to have a positive dimensional solution set as well as being a system that can be scaled up in size. The aforementioned mixed volume benchmark problems all lack positive dimensional solution sets, so we did not perform tests on them. We have found the cyclic n -roots problem to be the most interesting system that fulfills both criteria,

Table 3: Comparisons between Gfan and our implementation, for dimensions 3 through 8. Timings are listed in seconds.

n	Gfan	Pruning
3	0.036	0.12
4	0.23	0.25
5	2.03	0.80
6	23.49	10.73
7	299.32	49.53
8	3,764.83	540.32

as there are a variety of sizes of solution sets within them and the difficulty of computing pretropisms increases slowly. We also provide experimental data for the n -vortex and the n -body problem, but these problems quickly become uncomputable with our prototype Sage implementation.

5.1 Cyclic- n Experiments

The cyclic n -roots problem asks for the solutions of a polynomial system, commonly formulated as

$$\left\{ \begin{array}{l} x_0 + x_1 + \cdots + x_{n-1} = 0 \\ i = 2, 3, 4, \dots, n-1 : \sum_{j=0}^{n-1} \prod_{k=j}^{j+i-1} x_{k \bmod n} = 0 \\ x_0 x_1 x_2 \cdots x_{n-1} - 1 = 0. \end{array} \right. \quad (8)$$

This problem is important in the study of biunimodular vectors, a notion that traces back to Gauss, as stated in [19]. In [5], Backelin showed that if n has a divisor that is a square, i.e. if d^2 divides n for $d \geq 2$, then there are infinitely many cyclic n -roots. The conjecture of Björck and Saffari [8], [19, Conjecture 1.1] is that if n is not divisible by a square, then the set of cyclic n -roots is finite. As shown in [1], the result of Backelin can be recovered by polyhedral methods.

Instead of directly calculating the pretropisms of the Newton polytopes of the cyclic n -root problem, we chose to calculate pretropisms of the reduced cyclic n -root problem. This reformulation [14] is obtained by performing the substitution $x_i = \frac{y_i}{y_0}$ for $i = 0 \dots n-1$. Clearing the denominator of each equation leaves the first $n-1$ equations as polynomials in y_1, \dots, y_{n-1} . We compute pretropisms of the Newton polytopes of these $n-1$ equations because they yield meaningful sets of pretropisms. Calculating with the reduced cyclic n -roots problem has the benefit of removing much of the symmetry present in the standard cyclic n -roots problem, as well as decreasing the ambient dimension by one. Unlike the standard cyclic n -roots problem, some of the polytopes of the reduced cyclic n -roots problem are full dimensional, which leads to calculation

speed ups. A simple transformation can be performed on the pretropisms we calculate of reduced cyclic n -root problem to convert them to the pretropisms of cyclic n -root problem, so calculating the pretropisms of reduced cyclic n -roots problem is equivalent to calculating the pretropisms of the cyclic n -roots problem.

Table 4 shows how our implementation scales with time. As with the generic case, our implementation shows great improvement over the definitional algorithm as n becomes larger. For $n > 8$, the definitional algorithm was too inefficient to terminate in the time allotted.

Table 4: Comparisons between the definitional and our pruning method for reduced cyclic- n . Timings are listed in seconds.

n	Definitional	Pruning	#Pretropisms	Mixed Volume
4	0.02	0.62	2	4
5	0.43	1.04	0	14
6	17.90	1.56	8	26
7	301.26	2.57	28	132
8	33681.66	9.43	94	320
9		44.97	259	1224
10		978.67	712	3594

Just as we surpassed our estimates of the expected number of cone intersections in the generic case, we also surpassed our estimated ratio in the case of reduced cyclic- n . Table 5 contains experimental results.

Table 5: Number of cone intersections required for each algorithm, comparing the definitional algorithm with our pruning algorithm for reduced cyclic- n . The second to last column contains the ratio predicted by our cost estimate and the final column contains the actual ratio.

n	Definitional	Pruning	Predicted Ratio	Actual Ratio
4	120	44	0.25	0.36
5	1850	210	0.125	0.113
6	63,981	2,040	0.0625	0.0318
7	989,751	6,272	0.03125	0.00634
8	58,155,904	39,808	0.015625	0.000684
9		198,300	0.0078125	
10		1,933,147	0.00390625	

5.2 n -body and n -vortex

The n -body problem [23] is a classical problem from celestial dynamics that states that the acceleration due to Newtonian gravity can be found by solving

a system of equations (9). These equations can be turned into a polynomial system by clearing the denominators.

$$\ddot{x}_j = \sum_{i \neq j} \frac{m_i(x_i - x_j)}{r_{ij}^3} \quad 1 \leq j \leq n \quad (9)$$

The n -vortex problem [24] arose from a generalization of a problem from fluid dynamics that attempted to model vortex filaments (10). Again, these equations can be turned into polynomials through clearing denominators.

$$V_i = I \sum_{j \neq i} \frac{\Gamma_j}{z_i - z_j} \quad 1 \leq j \leq n \quad (10)$$

Table 6 displays experimental results for both the n -body problem and the n -vortex problem. We expect to be able to compute higher n for these benchmark problems when we develop a compiled version of this code.

Table 6: Experimental results of our new algorithm. Timings are in seconds. The last column gives the number of cone intersections.

System	n	Pruning Time	#Pretropisms	#Intersections
n -body	3	0.62	4	121
	4	5.07	57	25,379
	5	13,111.42	2,908	18,711,101
n -vortex	3	0.71	4	87
	4	2.93	25	10,595
	5	1457.48	569	5,021,659

6 Conclusion

To compute all pretropisms of a Laurent polynomial system, we propose to exploit the connectivity of edge skeletons to prune the tree of edges of the tuple of Newton polytopes. The horizontal and vertical pruning concepts we introduce are innovations that reduce the computational complexity of the problem. Our first high level implementation in Sage provides practical evidence that shows that our new pruning method is better than the definitional method with a variety of types of polynomial systems.

References

- [1] D. Adrovic and J. Verschelde. Computing Puiseux series for algebraic surfaces. In J. van der Hoeven and M. van Hoeij, editors, *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation (ISSAC 2012)*, pages 20–27. ACM, 2012.

- [2] D. Adrovic and J. Verschelde. Polyhedral methods for space curves exploiting symmetry applied to the cyclic n -roots problem. In V.P. Gerdt, W. Koepf, E.W. Mayr, and E.V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing, 15th International Workshop, CASC 2013, Berlin, Germany*, volume 8136 of *Lecture Notes in Computer Science*, pages 10–29, 2013.
- [3] B. Assarf, E. Gawrilow, K. Herr, M. Joswig, B. Lorenz, A. Paffenholz, and T. Rehn. Computing convex hulls and counting integer points with polymake. [arXiv:1408.4653v2](https://arxiv.org/abs/1408.4653v2).
- [4] D. Avis and K. Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete Comput. Geom.*, 8(3):295–313, 1992.
- [5] J. Backelin. Square multiples n give infinitely many cyclic n -roots. Reports, Matematiska Institutionen 8, Stockholms universitet, 1989.
- [6] R. Bagnara, P.M. Hill, and E. Zaffanella. The Parma Polyhedral Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1–2):3–21, 2008.
- [7] D.N. Bernshtein. The number of roots of a system of equations. *Functional Anal. Appl.*, 9(3):183–185, 1975.
- [8] G. Bjöck and B. Saffari. New classes of finite unimodular sequences with unimodular Fourier transforms. Circulant Hadamard matrices with complex entries. *C. R. Acad. Sci. Paris, Série I*, 320:319–324, 1995.
- [9] N. Bliss and J. Verschelde. Computing all space curve solutions of polynomial systems by polyhedral methods. Accepted for publication in the Proceedings of CASC 2016.
- [10] T. Bogart, A.N. Jensen, D. Speyer, B. Sturmfels, and R.R. Thomas. Computing tropical varieties. *Journal of Symbolic Computation*, 42(1):54–73, 2007.
- [11] V. Braun and M. Hampton. `polyhedra` module of Sage. The Sage Development Team, 2011.
- [12] B. Büeler, A. Enge, and K. Fukuda. Exact volume computation for polytopes: a practical study. In G. Kalai and G.M. Ziegler, editors, *Polytopes – Combinatorics and Computation*, volume 29 of *DMV Seminar*, pages 131–154. Springer-Verlag, 2000.
- [13] J.A. De Loera, J. Rambau, and F. Santos. *Triangulations. Structures for Algorithms and Applications*, volume 25 of *Algorithms and Computation in Mathematics*. Springer-Verlag, 2010.

- [14] I.Z. Emiris. *Sparse Elimination and Applications in Kinematics*. PhD thesis, University of California at Berkeley, Berkeley, 1994.
- [15] I.Z. Emiris and J.F. Canny. Efficient incremental algorithms for the sparse resultant and the mixed volume. *Journal of Symbolic Computation*, 20(2):117–149, 1995.
- [16] I.Z. Emiris and V. Fisikopoulos. Efficient random-walk methods for approximating polytope volume. In *Proceedings of the thirtieth annual symposium on computational geometry (SoCG’14)*, pages 318–327. ACM, 2014.
- [17] I.Z. Emiris, V. Fisikopoulos, and B. Gärtner. Efficient edge-skeleton computation for polytopes defined by oracles. *Journal of Symbolic Computation*, 73:139–152, 2016.
- [18] I.Z. Emiris, V. Fisikopoulos, and C. Konaxis. Exact and approximate algorithms for resultant polytopes. In *Proceedings of the 28th European Workshop on Computational Geometry (EuroCG12)*, 2012.
- [19] H. Führ and Z. Rzeszutnik. On biunimodular vectors for unitary matrices. *Linear Algebra and its Applications*, 484:86–129, 2015.
- [20] K. Fukuda and A. Prodon. Double description method revisited. In *Selected papers from the 8th Franco-Japanese and 4th Franco-Chinese Conference on Combinatorics and Computer Science*, volume 1120 of *Lecture Notes in Computer Science*, pages 91–111. Springer-Verlag, 1996.
- [21] T. Gao and T.Y. Li. Mixed volume computation for semi-mixed systems. *Discrete Comput. Geom.*, 29(2):257–277, 2003.
- [22] T. Gao, T.Y. Li, and M. Wu. Algorithm 846: MixedVol: a software package for mixed-volume computation. *ACM Trans. Math. Softw.*, 31(4):555–560, 2005.
- [23] M. Hampton and A. Jensen. Finiteness of relative equilibria in the planar generalized n-body problem with fixed subconfigurations. *Journal of Geometric Mechanics*, 7(1):35–42, 2015.
- [24] M. Hampton and R. Moeckel. Finiteness of stationary configurations of the four-vortex problem. *Transactions of the American Mathematical Society*, 361(3):1317–1332, 2009.
- [25] A.N. Jensen. Gfan, a software system for Gröbner fans and tropical varieties. Available at <http://home.imf.au.dk/jensen/software/gfan/gfan.html>.
- [26] A.N. Jensen. Computing Gröbner fans and tropical varieties in Gfan. In M.E. Stillman, N. Takayama, and J. Verschelde, editors, *Software for Algebraic Geometry*, volume 148 of *The IMA Volumes in Mathematics and its Applications*, pages 33–46. Springer-Verlag, 2008.

- [27] D. Maclagan and B. Sturmfels. *Introduction to Tropical Geometry*, volume 161 of *Graduate Studies in Mathematics*. American Mathematical Society, 2015.
- [28] G. Malajovich. Computing mixed volume and all mixed cells in quermass-integral time. To appear in *Found. Comput. Math.*
- [29] J. Maurer. Puiseux expansion for space curves. *Manuscripta Math.*, 32:91–100, 1980.
- [30] T. Mizutani and A. Takeda. DEMiCs: a software package for computing the mixed volume via dynamic enumeration of all mixed cells. In M.E. Stillman, N. Takayama, and J. Verschelde, editors, *Software for Algebraic Geometry*, volume 148 of *The IMA Volumes in Mathematics and Its Applications*, pages 59–79. Springer-Verlag, 2008.
- [31] T. Mizutani, A. Takeda, and M. Kojima. Dynamic enumeration of all mixed cells. *Discrete Comput. Geom.*, 37(3):351–367, 2007.
- [32] A.Y. Novoseltsev. `lattice_polytope` module of Sage. The Sage Development Team, 2011.
- [33] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.
- [34] J. Sommars and J. Verschelde. Exact gift wrapping to prune the tree of edges of Newton polytopes to compute pretropisms. [arXiv:1512.01594](https://arxiv.org/abs/1512.01594).
- [35] J. Sommars and J. Verschelde. Computing pretropisms for the cyclic n-roots problem. In *32nd European Workshop on Computational Geometry (EuroCG'16)*, pages 235–238, 2016.
- [36] W.A. Stein et al. *Sage Mathematics Software (Version 6.9)*. The Sage Development Team, 2015. <http://www.sagemath.org>.
- [37] J. Verschelde. Algorithm 795: PHCpack: A general-purpose solver for polynomial systems by homotopy continuation. *ACM Trans. Math. Softw.*, 25(2):251–276, 1999.
- [38] J. Verschelde. Polyhedral methods in numerical algebraic geometry. In D.J. Bates, G. Besana, S. Di Rocco, and C.W. Wampler, editors, *Interactions of Classical and Numerical Algebraic Geometry*, volume 496 of *Contemporary Mathematics*, pages 243–263. AMS, 2009.
- [39] G.M. Ziegler. *Lectures on Polytopes*, volume 152 of *Graduate Texts in Mathematics*. Springer-Verlag, 1995.

Algorithm 1 Explores the skeleton of edges to find pretropisms of a polytope and a cone.

1: **function** EXPLOREEDGESKELETON(Polytope P , Cone C)

2: r := a random ray inside C

3: $\text{in}_r(P)$:= vertices of P with minimal inner product with r

4: EdgesToTest := all edges of P that have vertices in $\text{in}_r(P)$

5: Cones := \emptyset

6: TestedEdges := \emptyset

7: **while** EdgesToTest $\neq \emptyset$ **do**

8: E := pop an edge from EdgesToTest

9: C_E := normal cone to E

10: ShouldAddCone := False

11: **if** C_E contains C **then**

12: ConeToAdd := C

13: ShouldAddCone := True

14: **else if** $C \cap C_E \neq \{0\}$ **then**

15: ConeToAdd := $C \cap C_E$

16: ShouldAddCone := True

17: **end if**

18: **if** ShouldAddCone **then**

19: Cones := Cones \cup ConeToAdd

20: Edges := Edges $\cup E$

21: **for** each neighboring edge e of E **do**

22: **if** $e \notin$ TestedEdges **then**

23: EdgesToTest := EdgesToTest $\cup e$

24: **end if**

25: **end for**

26: **end if**

27: TestedEdges := TestedEdges $\cup E$

28: **end while**

29: **return** Cones

30: **end function**

Algorithm 2 Finds pretropisms for a given set of polytopes

```

1: function FINDPRETROPISMS(Polytope  $P_1$ , Polytope  $P_2, \dots$ , Polytope  $P_n$ )
2:   Cones := set of normal cones to edges in  $P_1$ 
3:   for  $i := 2$  to  $n$  do
4:     NewCones :=  $\emptyset$ 
5:     for Cone in Cones do
6:       NewCones := NewCones  $\cup$  ExploreEdgeSkeleton( $P_i$ , Cone)
7:     end for
8:     for Cone in NewCones do
9:       if Cone is contained within another cone in NewCones then
10:        NewCones := NewCones - Cone
11:       end if
12:     end for
13:     Cones := NewCones
14:   end for
15:   Pretropisms := set of generating rays for each cone in Cones
16:   return Pretropisms
17: end function

```
